I/O – where OS meet the real world

David E. Culler

CS162 – Operating Systems and Systems Programming

Lecture 21

October 17, 2014

Reading: A&D 11.3, 12
HW 4 out
Proj 2 out

What is the Role of I/O?

- Without I/O, computers are useless (disembodied brains?)
- But… thousands of devices, each slightly different
  - How can we standardize the interfaces to these devices?
- Devices unpredictable and/or slow
  - How can we manage them if we don't know what they will do or how they will perform?
- Devices unreliable: media failures and transmission errors
  - How can we make them reliable???

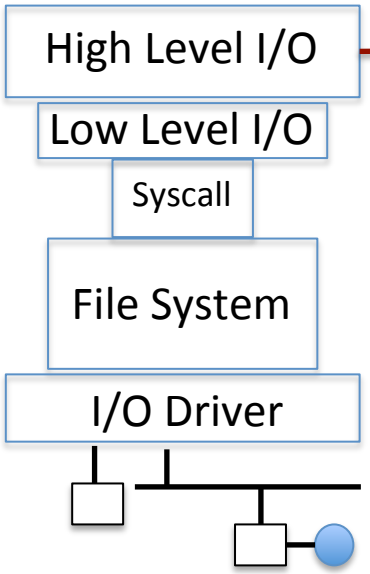
Objectives

- Understand concept of I/O driver
  - OS module that interfaces to particular device and provides a consistent I/O interface
    - Still quite low level compared to user API
- Understand Basic Classes of Devices / Drivers
- Build a simple performance model to guide thinking
- Understand storage devices below the file system

Recall: I/O & Storage Layers

Operations, Entities and Interface

Application / Service



streams — fopen, fread, fgets, …, fprintf, … on FILE *

format, buffer streams, call low I/O

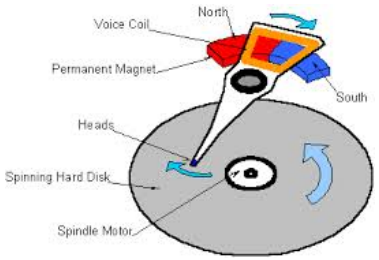
handles — open, read, write, close on ints & void *

registers

descriptors

Commands and Data Transfers

Disks, Flash, Controllers, DMA



I/O & Storage Layers

Operations, Entities and Interface

Application / Service

High Level I/O

Low Level I/O

Syscall

File System

I/O Driver

streams

handles

registers

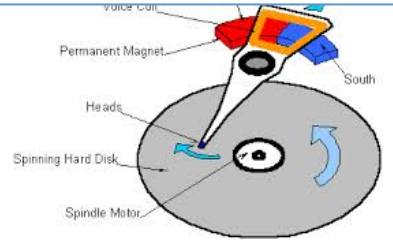
open, read, write, close on ints & void *

Move args to regs, issue syscalls

EAX, EBX, … ESP

pintos/src/lib/user/syscall.c

```
int
read (int fd, void *buffer, unsigned size)
{
  return syscall3 (SYS_READ, fd, buffer, size);
}
```



I/O & Storage Layers

Operations, Entities and Interface

Application / Service

High Level I/O — *streams*

Low Level I/O — *handles*

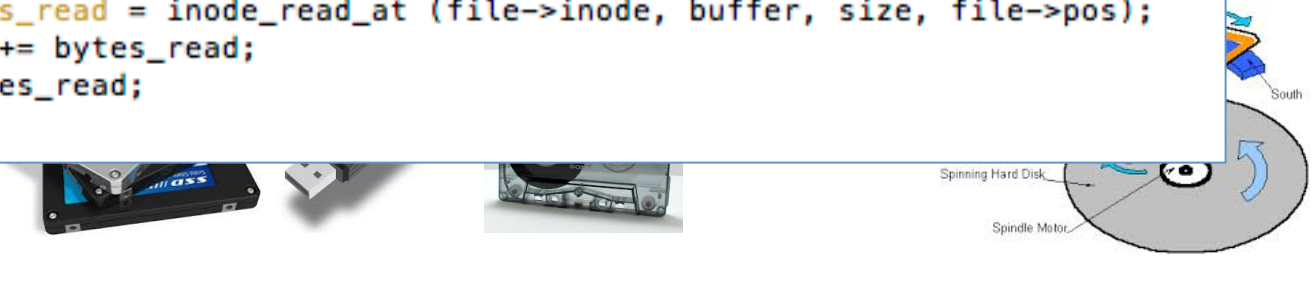
Proj 2 — Syscall — *registers* — EAX, EBX, … ESP

Regs => Args, Dispatch call to Handler

File System — *descriptors* — file_open, file_read, … on **struct file *** & void *

pintos/src/filesys/file.c — *Commands and Data Transfers*

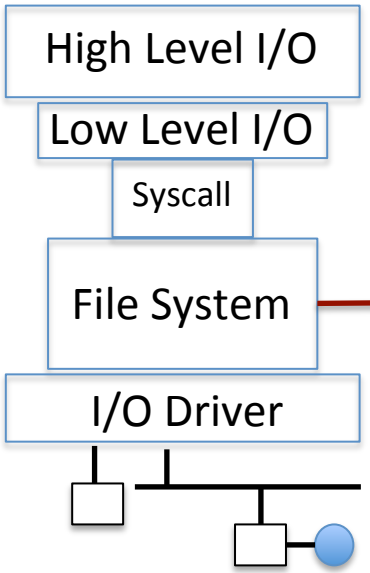
```
   Advances FILE's position by the number of bytes read. */
off_t
file_read (struct file *file, void *buffer, off_t size)
{
  off_t bytes_read = inode_read_at (file->inode, buffer, size, file->pos);
  file->pos += bytes_read;
  return bytes_read;
}
```



I/O & Storage Layers

Operations, Entities and Interface

Application / Service



streams

handles

registers

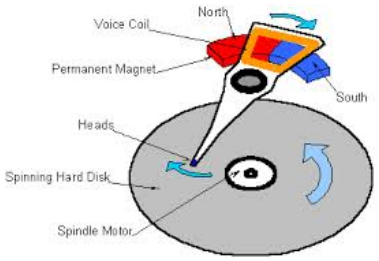
file_open, file_read, … on **struct file *** & void *

descriptors

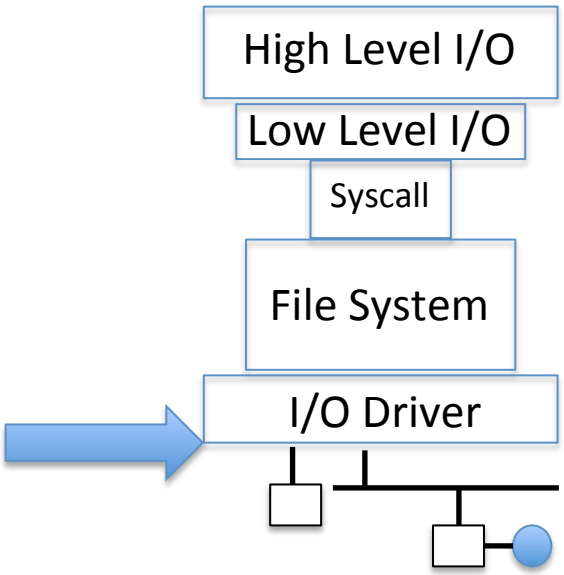
Next Week

Commands and Data Transfers

Disks, Flash, Controllers, DMA



I/O & Storage Layers – Today



Operations and Interface

streams fopen, fread, fgets, …, fwrite, fclose on FILE *

handles open, read, write, close on int & char *

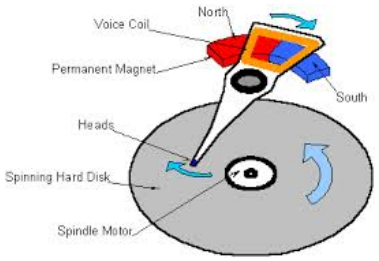
registers EAX, EBX, … ESP

descriptors

Commands and Data Transfers

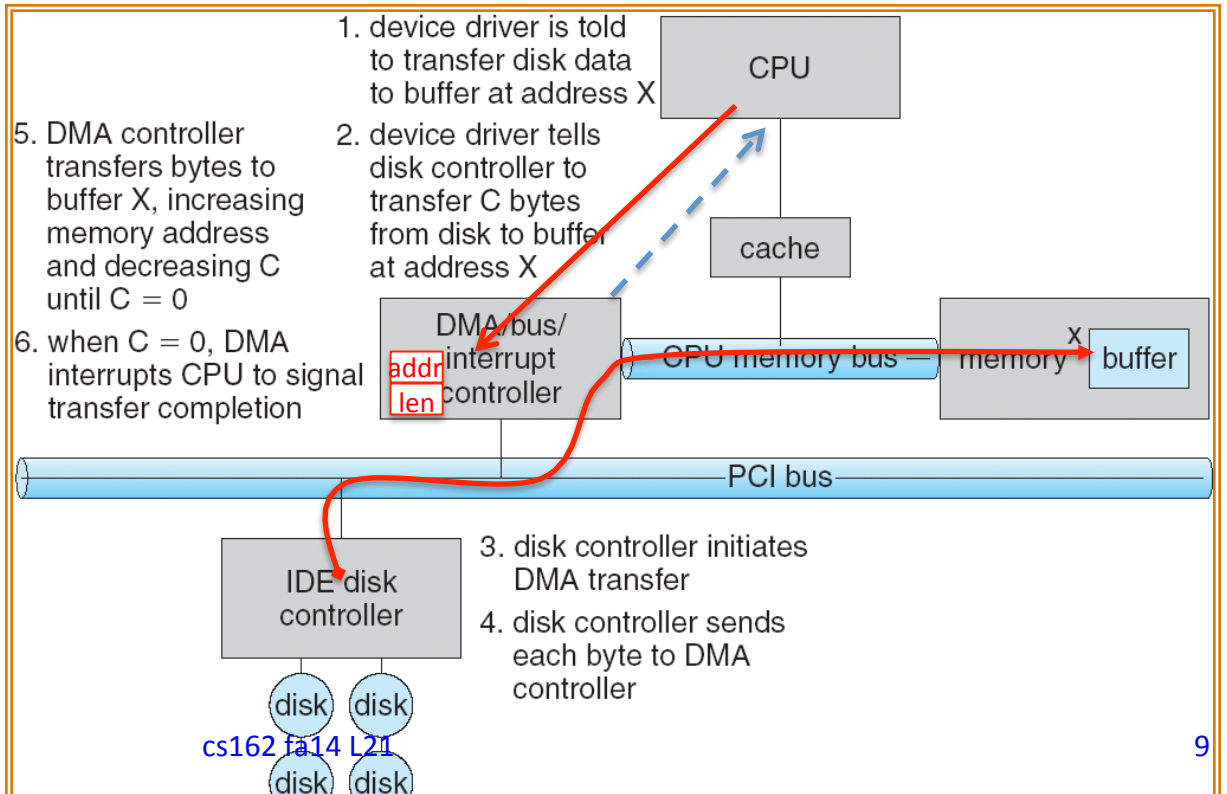
ld, st PIO ctrl regs, dma

Disks, Flash, Controllers, DMA



Transferring Data To/From Controller

- Programmed I/O:
  - Each byte transferred via processor in/out or load/store
  - Pro: Simple hardware, easy to program
  - Con: Consumes processor cycles proportional to data size
- Direct Memory Access:
  - Give controller access to memory bus
  - Ask it to transfer data blocks to/from memory directly
- Sample interaction with DMA controller (from OSC):



I/O Device Notifying the OS

- The OS needs to know when:
  - The I/O device has completed an operation
  - The I/O operation has encountered an error
- I/O Interrupt:
  - Device generates an interrupt whenever it needs service
  - Pro: handles unpredictable events well
  - Con: interrupts relatively high overhead
- Polling:
  - OS periodically checks a device-specific status register
    - I/O device puts completion information in status register
  - Pro: low overhead
  - Con: may waste many cycles on polling if infrequent or unpredictable I/O operations
- Actual devices combine both polling and interrupts
  - For instance – High-bandwidth network adapter:
    - Interrupt for first incoming packet
    - Poll for following packets until hardware queues are empty

Operational Parameters for I/O

- *Data granularity:* Byte vs. Block
  - Some devices provide single byte at a time (*e.g.,* keyboard)
  - Others provide whole blocks (*e.g.,* disks, networks, etc.)
- *Access pattem*: Sequential vs. Random
  - Some devices must be accessed sequentially (*e.g.,* tape)
  - Others can be accessed "randomly" (*e.g.,* disk, cd, etc.)
    - Fixed overhead to start sequential transfer (more later)
- *Transfer Notification:* Polling vs. Interrupts
  - Some devices require continual monitoring
  - Others generate interrupts when they need service
- *Transfer Mechanism:* Programmed IO and DMA

The Goal of the I/O Subsystem

- Provide uniform interfaces, despite wide range of different devices
  - This code works on many different devices:

```
FILE *fd = fopen("/dev/something","rw");
for (int i = 0; i < 10; i++) {
   fprintf(fd, "Count %d\n",i);
}
close(fd);
```

  - Why?
  - Because code that controls devices ("device driver") implements standard interface
  - Standard user library raise the standard driver / subsystem interface
- We will try to get a flavor for what is involved in actually controlling devices
  - Can only scratch surface!

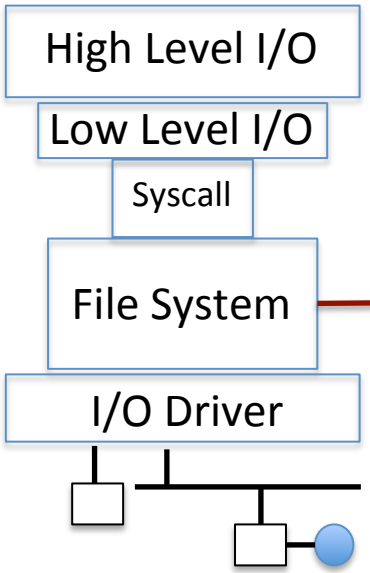
Want Standard Interfaces to Devices

- Block Devices: *e.g.,* disk drives, tape drives, DVD-ROM
  - Access blocks of data
  - Driver Commands include `open()`, `read()`, `write()`, `seek()`
  - Raw I/O or file-system access
  - Memory-mapped file access possible (discussed later … VAS!)
- Character/Byte Devices: *e.g.,* keyboards, mice, serial ports, some USB devices
  - Single characters at a time
  - Commands include `get()`, `put()`
  - Libraries layered on top allow line editing
- Network Devices: *e.g.,* Ethernet, Wireless, Bluetooth
  - Different enough from block/character to have own interface
  - Unix and Windows include socket interface
    - Separates network protocol from network operation
    - Includes `select()` functionality

I/O & Storage Layers

Operations, Entities and Interface

Application / Service



streams

handles

registers

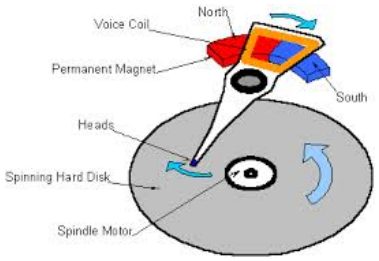
file_open, file_read, … on **struct file *** & void *

descriptors

open, read, … on **???** (blocks, chars, nets)

Commands and Data Transfers

Disks, Flash, Controllers, DMA

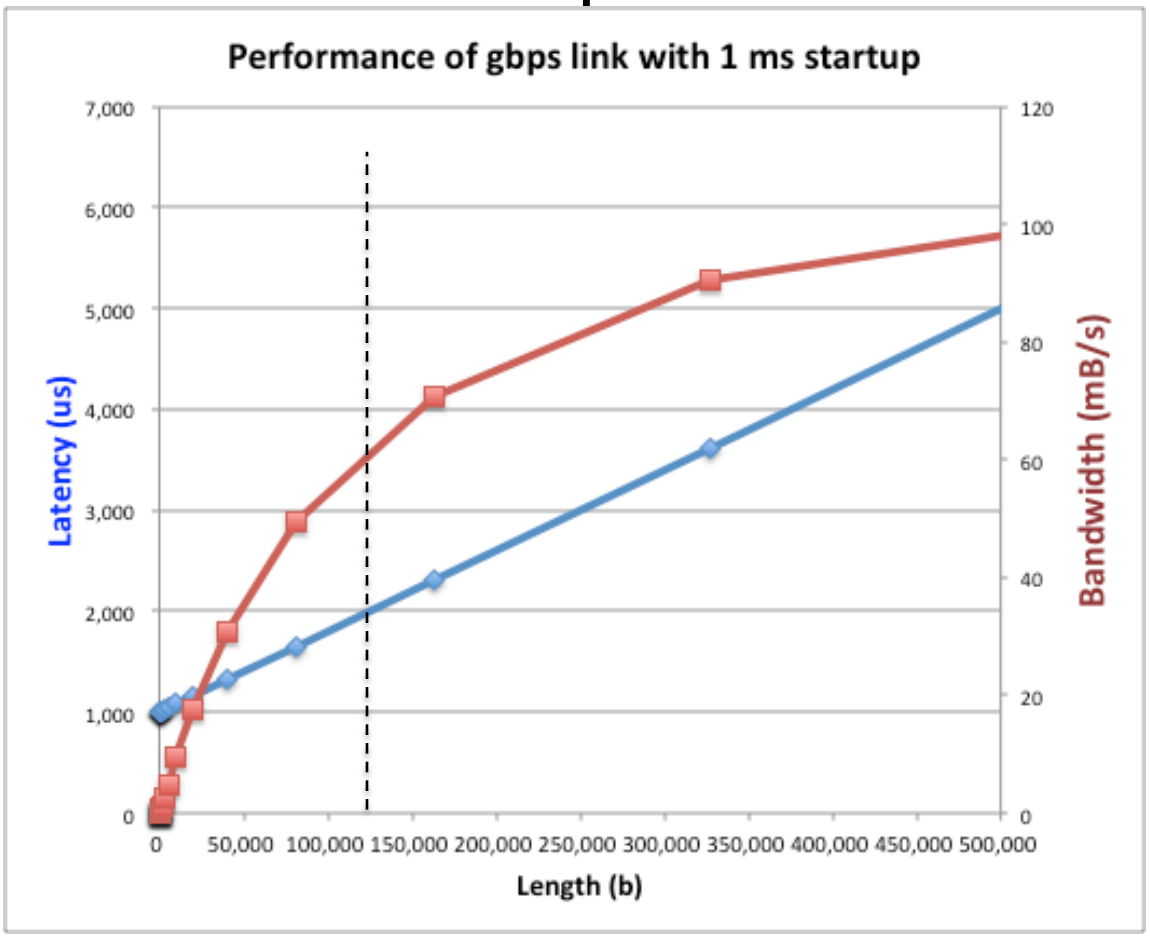


Basic Performance Concepts

- *Response Time* or *Latency*: Time to perform an operation (s)
- *Bandwidth* or *Throughput*: Rate at which operations are performed (op/s)
  - Files: mB/s, Networks: mb/s, Arithmetic: GFLOP/s
- *Start up* or "Overhead": time to initiate an operation
- Most I/O operations are roughly linear
  - Latency (n) = Ovhd + n/Bandwdth

Example (fast network)

- Consider a gpbs link (125 mB/s)
- With a startup cost S = 1 ms

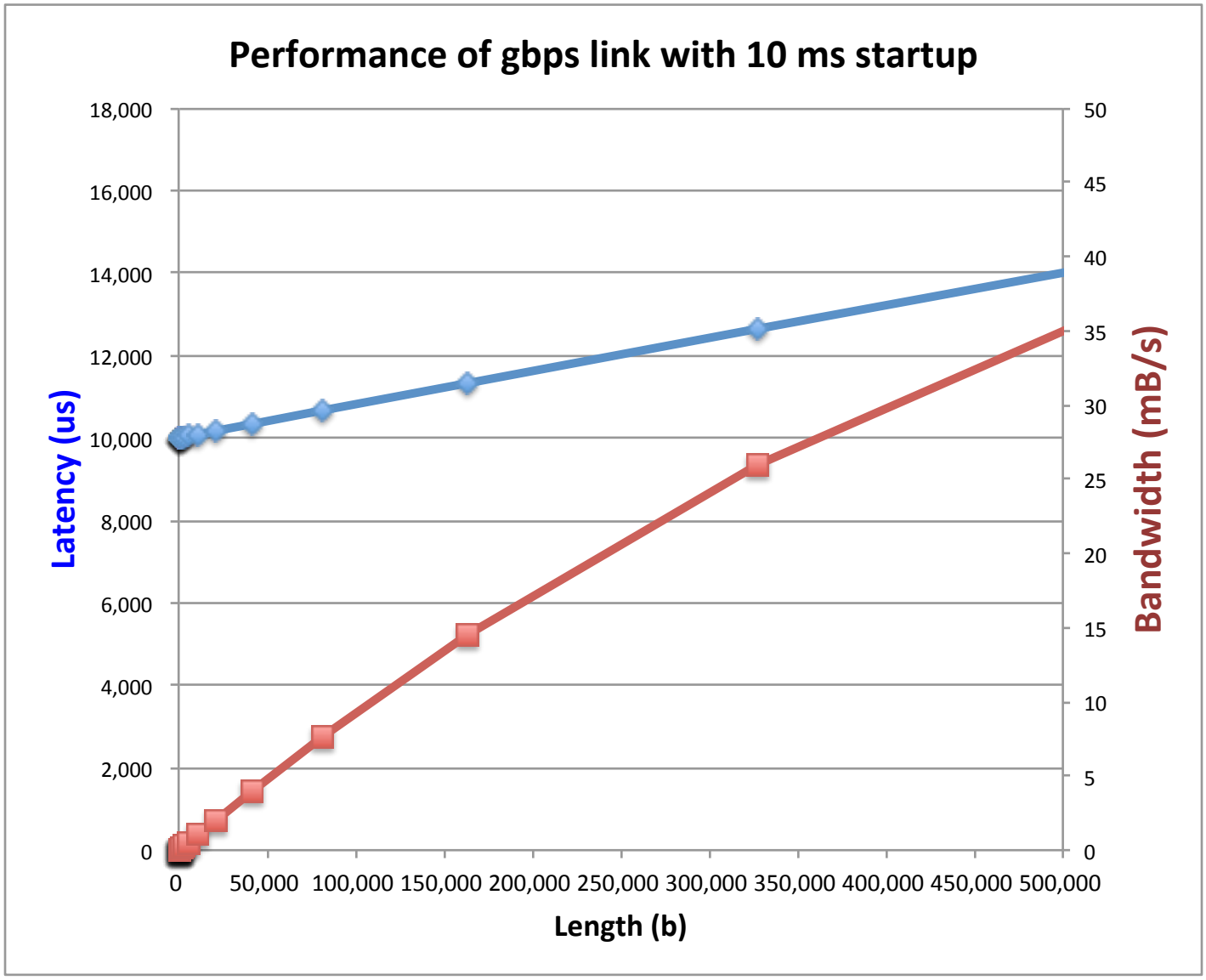


Theorem: half-power point occurs at n=S*B

- When transfer time = startup

T(S*B) = S + S*B/B

Example: at 10 ms startup (disk)



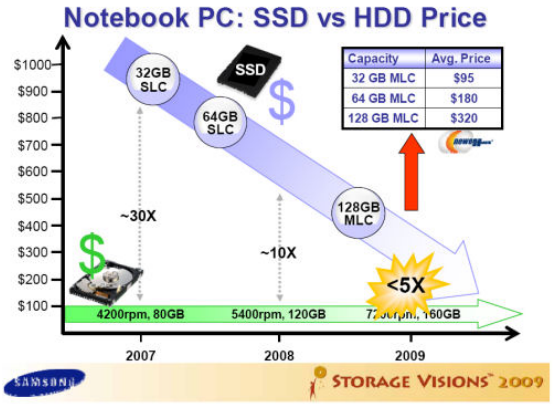
What determines peak BW for I/O ?

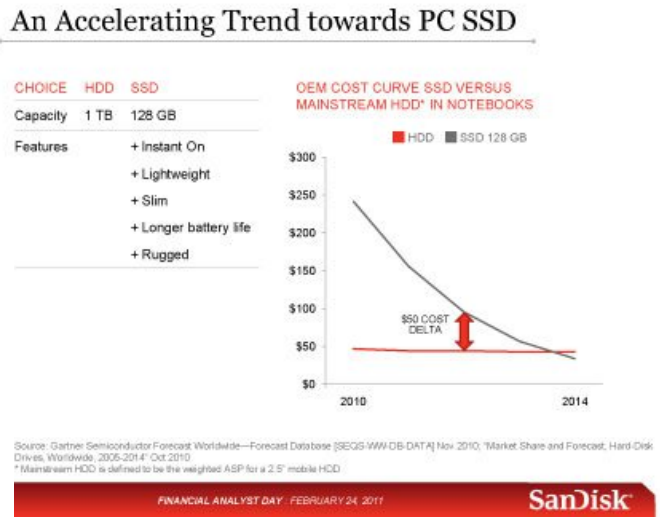
- Bus Speed
  - PCI-X: 1064 MB/s = 133 MHz x 64 bit (per lane)
  - ULTRA WIDE SCSI: 40 MB/s
  - Serial Attached SCSI & Serial ATA & IEEE 1394 (firewire) : 1.6 Gbps full duplex (200 MB/s)
  - USB 1.5 – 12 mb/s
- Device Transfer Bandwidth
  - Rotational speed of disk
  - Write / Read rate of nand flash
  - Signaling rate of network link
- Whatever is the bottleneck in the path

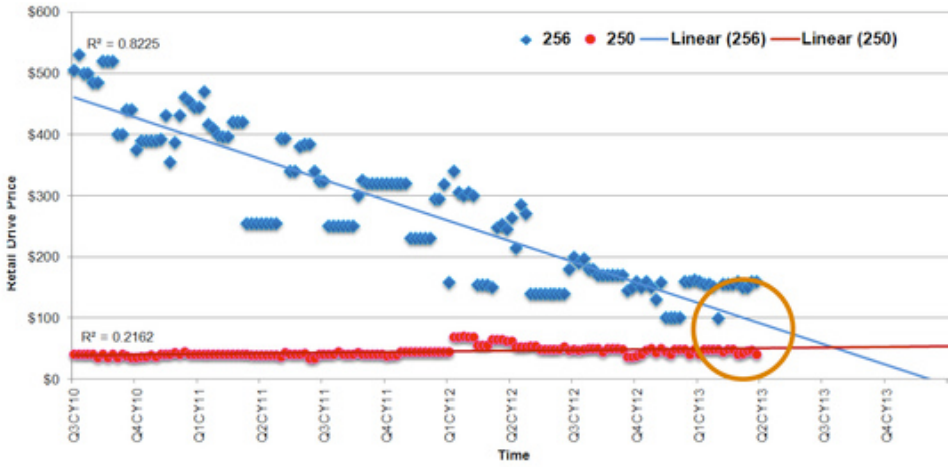
Storage Devices

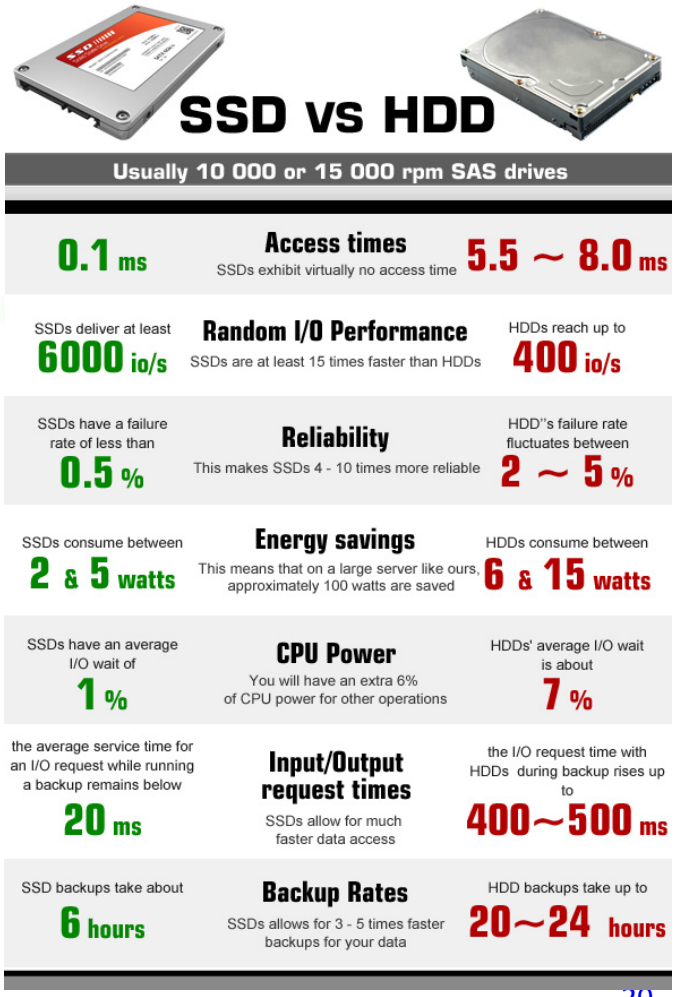
- Magnetic disks
  - Storage that rarely becomes corrupted
  - Large capacity at low cost
  - Block level random access
  - Slow performance for random access
  - Better performance for streaming access
- Flash memory
  - Storage that rarely becomes corrupted
  - Capacity at intermediate cost (50x disk ???)
  - Block level random access
  - Good performance for reads; worse for random writes
  - Erasure requirement in large blocks
  - Wear patterns

Are we in an inflection point?

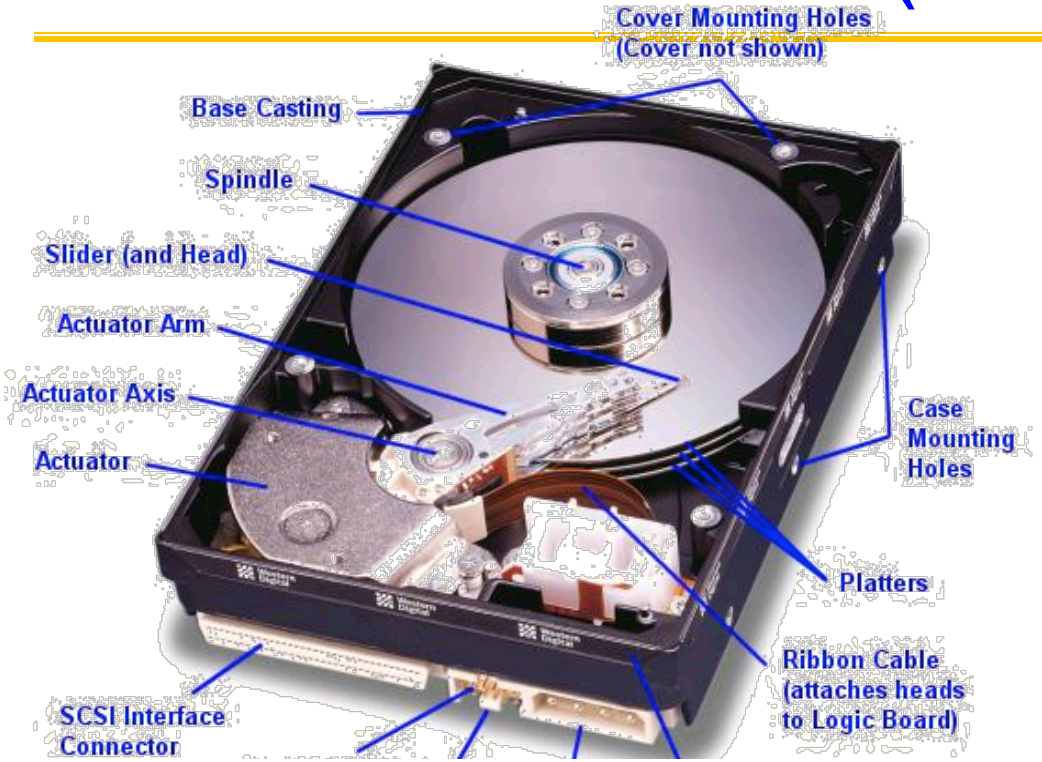




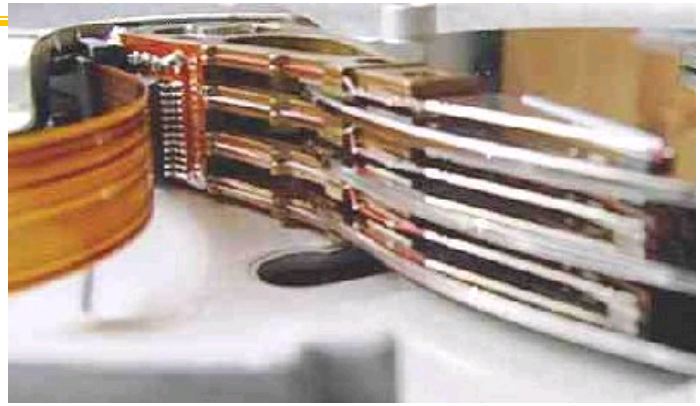




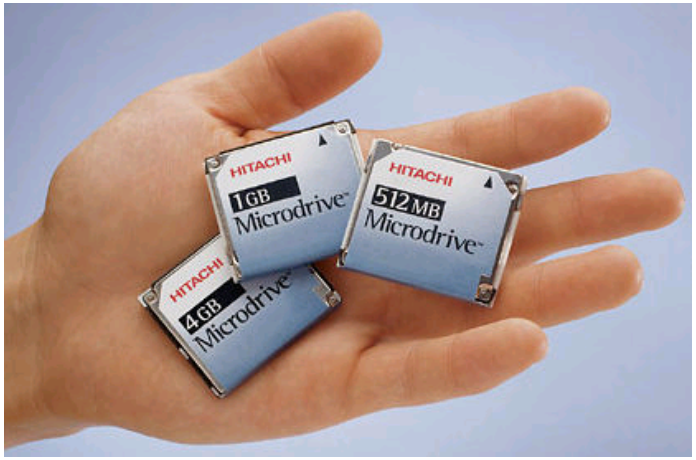
Hard Disk Drives (HDDs)



Western Digital Drive
http://www.storagereview.com/guide/



Read/Write Head Side View

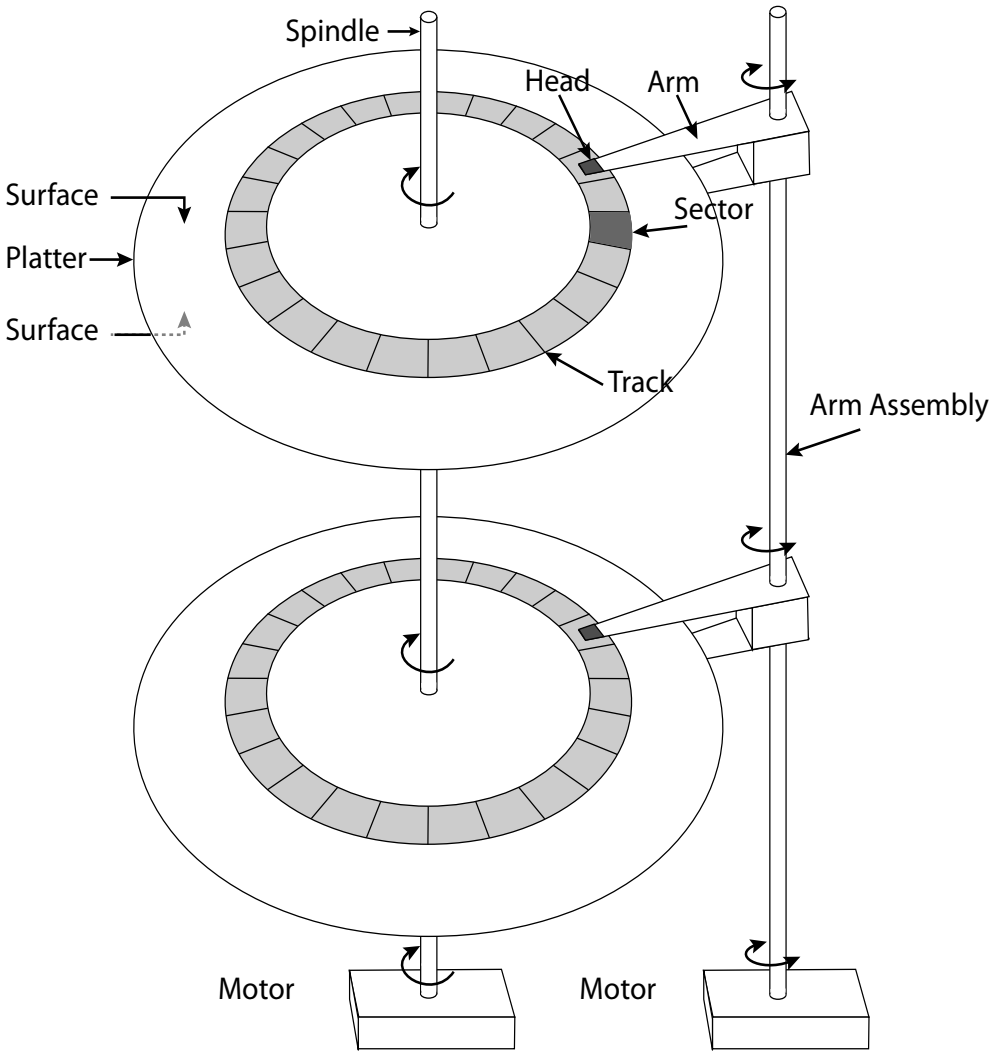


IBM/Hitachi Microdrive

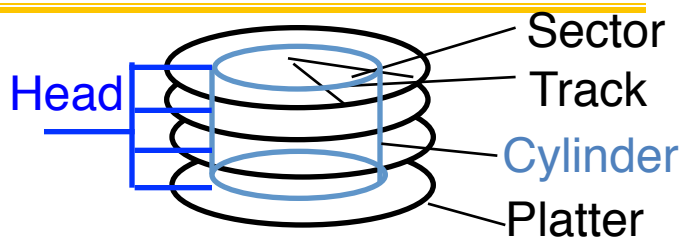
IBM Personal Computer/AT (1986)
- 30 MB hard disk - $500
- 30-40ms seek time
- 0.7-1 MB/s (est.)

The Amazing Magnetic Disk

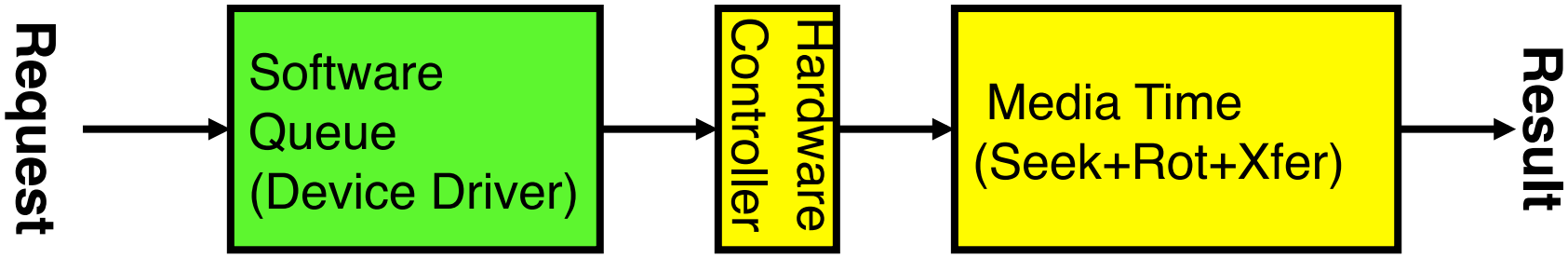
- Unit of Transfer: Sector
  - Ring of sectors form a track
  - Stack of tracks form a cylinder
  - Heads position on cylinders
- Disk Tracks ~ 1 micron wide
  - Wavelength of light is ~ 0.5 micron
  - Resolution of human eye: 50 microns
  - 100K on a typical 2.5" disk
- Separated by unused guard regions
  - Reduces likelihood neighboring tracks are corrupted during writes (still a small non-zero chance)
- Track length varies across disk
  - Outside: More sectors per track, higher bandwidth
  - Disk is organized into regions of tracks with same # of sectors/track
  - Only outer half of radius is used
    - Most of the disk area in the outer regions of the disk



Magnetic Disk Characteristic



- Cylinder: all the tracks under the head at a given point on all surfaces
- Read/write: three-stage process:
  - **Seek time**: position the head/arm over the proper track (into proper cylinder)
  - **Rotational latency**: wait for the desired sector to rotate under the read/write head
  - **Transfer time**: transfer a block of bits (sector) under the read-write head
- Disk Latency = Queuing Time + Controller time + Seek Time + Rotation Time + Xfer Time



- Highest Bandwidth:
  - Transfer large group of blocks sequentially from one track

Typical Numbers for Magnetic Disk

Parameter	Info / Range
Average seek time	**Typically 5-10 milliseconds**. Depending on reference locality, actual cost may be 25-33% of this number.
Average rotational latency	Most laptop/desktop disks rotate at 3600-7200 RPM (16-8 ms/rotation). Server disks up to 15,000 RPM. Average latency is halfway around disk yielding corresponding times of **8-4 milliseconds**
Controller time	Depends on controller hardware
Transfer time	**Typically 50 to 100 MB/s**. Depends on: • Transfer size (usually a sector): 512B – 1KB per sector • Rotation speed: 3600 RPM to 15000 RPM • Recording density: bits per inch on a track • Diameter: ranges from 1 in to 5.25 in
Cost	Drops by a factor of two every 1.5 years (or even faster). **$0.03-0.07/GB in 2013**

Intelligence in the controller

Sectors contain sophisticated error correcting codes
  - Disk head magnet has a field wider than track
  - Hide corruptions due to neighboring track writes
- Sector sparing
  - Remap bad sectors transparently to spare sectors on the same surface
- Slip sparing
  - Remap all sectors (when there is a bad sector) to preserve sequential behavior
- Track skewing
  - Sector numbers offset from one track to the next, to allow for disk head movement for sequential ops
- …

Question

- How long to complete 500 random disk reads, in FIFO order?

Question

- How long to complete 500 random disk reads, in FIFO order?
  - Seek: average 10.5 msec
  - Rotation: average 4.15 msec
  - Transfer: 5-10 usec
- 500 * (10.5 + 4.15 + 0.01)/1000 = 7.3 seconds

Question

- How long to complete 500 sequential disk reads?

Question

- How long to complete 500 sequential disk reads?
  - Seek Time: 10.5 ms (to reach first sector)
  - Rotation Time: 4.15 ms (to reach first sector)
  - Transfer Time: (outer track)

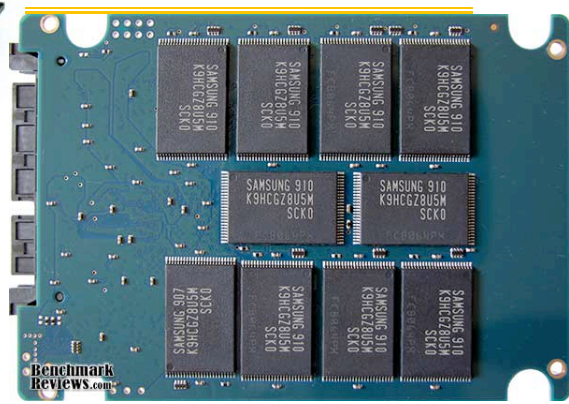
    500 sectors * 512 bytes / 128MB/sec = 2ms

Total: 10.5 + 4.15 + 2 = 16.7 ms

Might need an extra head or track switch (+1ms)

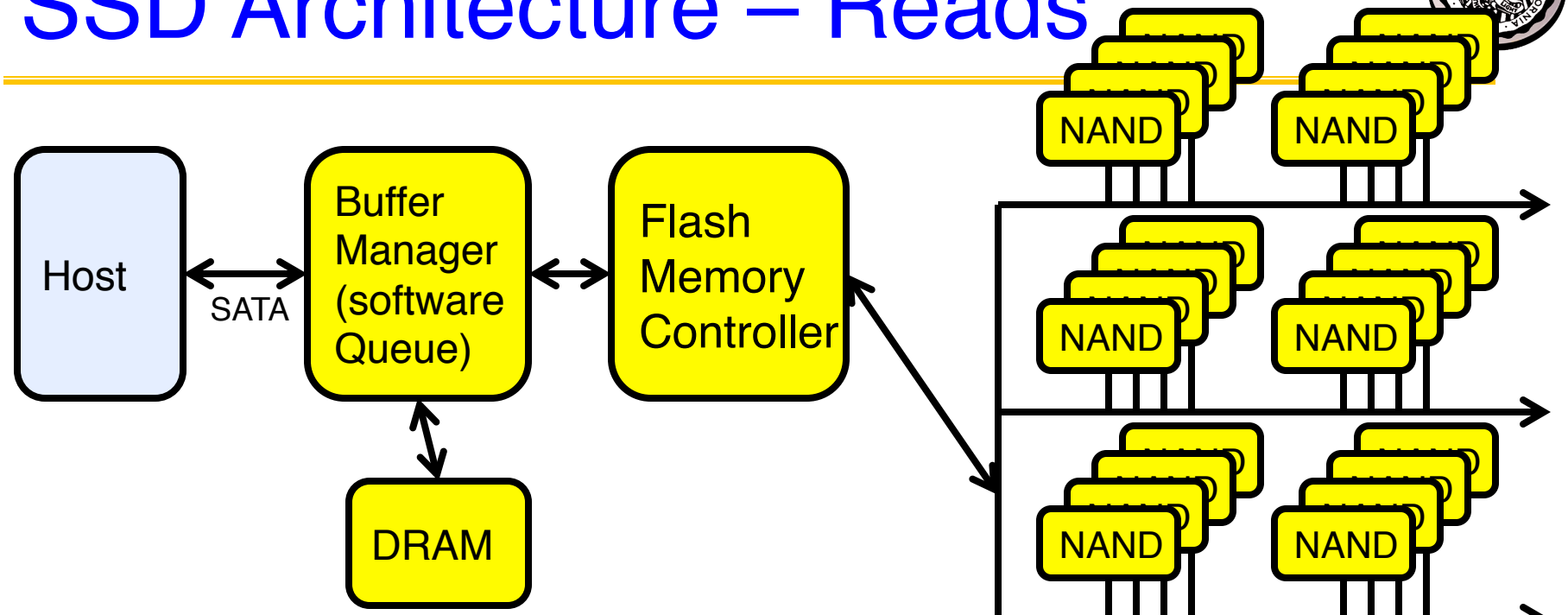
Track buffer may allow some sectors to be read off disk out of order (-2ms)

Solid State Disks (SSDs)



- 1995 – Replace rotating magnetic media with non-volatile memory (battery backed DRAM)
- 2009 – Use NAND Multi-Level Cell (2-bit/cell) flash memory
  - Sector (4 KB page) addressable, but stores 4-64 “pages” per memory block
- No moving parts (no rotate/seek motors)
  - Eliminates seek and rotational delay (0.1-0.2ms access time)
  - Very low power and lightweight
  - Limited “write cycles”
- Rapid advance in capacity and cost since

SSD Architecture – Reads

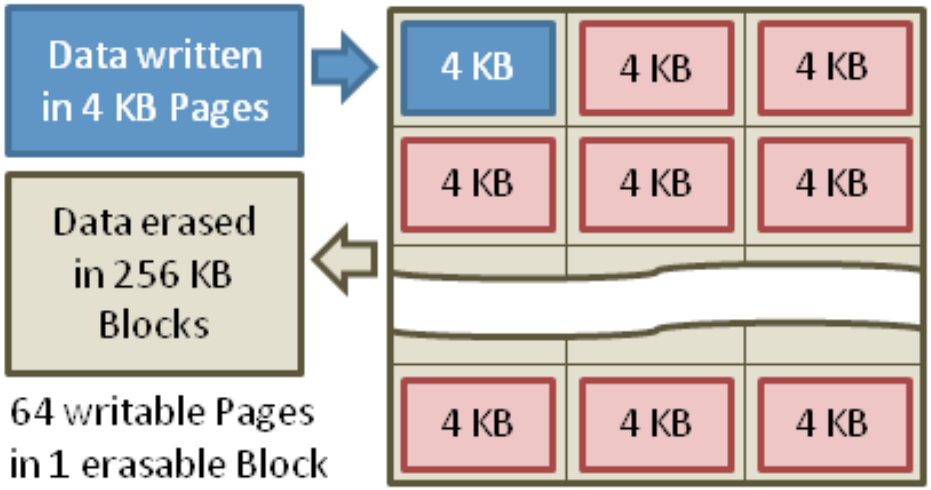


Read 4 KB Page: ~25 usec

- No seek or rotational latency
- Transfer time: transfer a 4KB page
  - SATA: 300-600MB/s => ~4 x10^3 b / 400 x 10^6 bps => 10 us
- Latency = Queuing Time + Controller time + Xfer Time
- Highest Bandwidth: Sequential OR Random reads

SSD Architecture – Writes (I)

- Writing data is complex! (~200µs – 1.7ms)
  - Can only write empty pages in a block
  - Erasing a block takes ~1.5ms
  - Controller maintains pool of empty blocks by coalescing used pages (read, erase, write), also reserves some % of capacity
- Rule of thumb: writes 10x reads, erasure 10x writes



https://en.wikipedia.org/wiki/Solid-state_drive

Storage Performance & Price (jan 13)

	Bandwidth (Sequential R/W)	Cost/GB	Size
HDD[2]	50-100 MB/s	$0.03-0.07/GB	2-4 TB
SSD[1,2]	200-550 MB/s (SATA) 6 GB/s (read PCI) 4.4 GB/s (write PCI)	$0.87-1.13/GB	200GB-1TB
DRAM[2]	10-16 GB/s	$4-14*/GB *SK Hynix 9/4/13 fire	64GB-256GB

[1] http://www.fastestssd.com/featured/ssd-rankings-the-fastest-solid-state-drives/

[2] http://www.extremetech.com/computing/164677-storage-pricewatch-hard-drive-and-ssd-prices-drop-making-for-a-good-time-to-buy

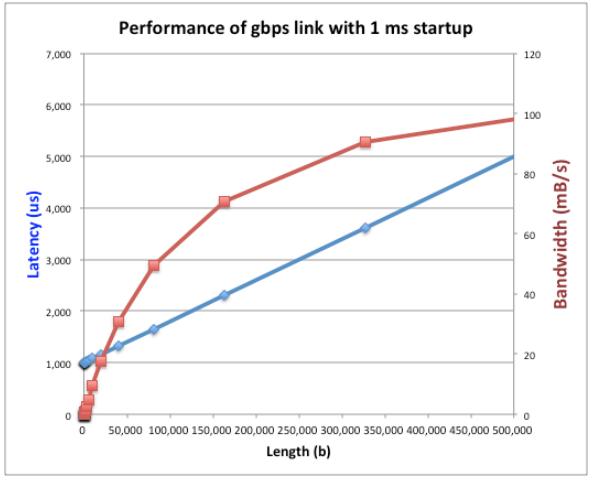
BW: SSD up to x10 than HDD, DRAM > x10 than SSD
Price: HDD x20 less than SSD, SSD x5 less than DRAM

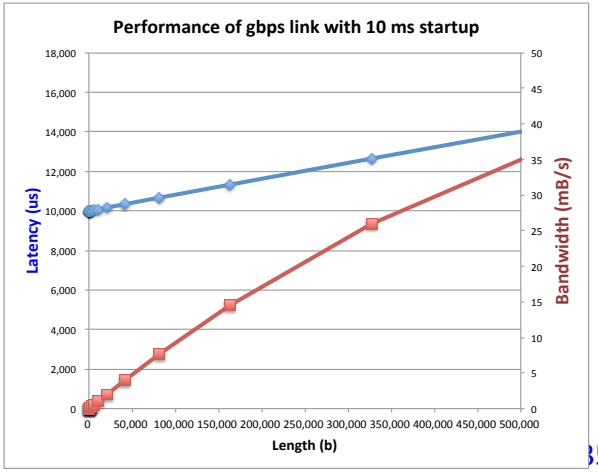
SSD Summary

- Pros (vs. hard disk drives):
  - Low latency, high throughput (eliminate seek/rotational delay)
  - No moving parts:
    - Very light weight, low power, silent, very shock insensitive
  - Read at memory speeds (limited by controller and I/O bus)
- Cons
  - Small storage (0.1-0.5x disk), expensive (20x disk ???)
    - Hybrid alternative: combine small SSD with large HDD
  - Asymmetric block write performance: read pg/erase/write pg
    - Controller garbage collection (GC) algorithms have major effect on performance
  - Limited drive lifetime
    - 1-10K writes/page for MLC NAND
    - Avg failure rate is 6 years, life expectancy is 9–11 years
- These are changing rapidly

What goes into startup cost for I/O?

- Syscall overhead
- Operating system processing
- Controller Overhead
- Device Startup
  - Mechanical latency for a disk
  - Media Access + Speed of light + Routing for network
- Queuing (next week)





Summary

- Drivers interface to I/O devices
  - Provide clean Read/Write interface to OS above
  - Manipulate devices through PIO, DMA & interrupt handling
  - 2 types: block, character, and network
- Devices have complex protocols for interaction and performance characteristics
  - Response time (Latency) = Queue + Overhead + Transfer
    - Effective BW = BW * T/(S+T)
  - HDD: controller + seek + rotation + transfer
  - SDD: controller + transfer (erasure & wear)
- Bursts & High Utilization introduce queuing delays
- Systems (e.g., file system) designed to optimize performance and reliability
  - Relative to performance characteristics of underlying device